Making the Most out of Direct-Access Network Attached Storage

Making the Most out of Direct-Access Network Attached Storage

Kostas Magoutis, Salimah Addetia, Alexandra Fedorova, Margo I. Seltzer
Division of Engineering and Applied Sciences, Harvard University

Abstract

The performance of high-speed network-attached storage applications is often limited by end-system overhead, caused primarily by memory copying and network protocol processing. In this paper, we examine alternative strategies for reducing overhead in such systems. We consider optimizations to remote procedure call (RPC)-based data transfer using either remote direct memory access (RDMA) or network interface support for pre-posting of application receive buffers. We demonstrate that both mechanisms enable file access throughput that saturates a 2Gb/s network link when performing large I/Os on relatively slow, commodity PCs. However, for multi-client workloads dominated by small I/Os, throughput is limited by the per-I/O overhead of processing RPCs in the server. For such workloads, we propose the use of a new network I/O mechanism, *Optimistic RDMA (ORDMA)*. ORDMA is an alternative to RPC that aims to improve server throughput and response time for small I/Os. We measured performance improvements of up to 32% in server throughput and 36% in response time with use of ORDMA in our prototype.

1 Introduction

The performance of I/O-intensive applications using network-attached storage (NAS) systems over high-speed networks is often associated with high CPU and memory system overhead [3,6,9,12,20,23,29,30]. This overhead is primarily due to unnecessary memory copying and transport protocol processing, caused by inefficiencies in transporting file I/O traffic over general-purpose network protocol stacks. Memory copying is a per-byte source of overhead that limits the I/O bus throughput available for network transfers. Protocol processing, however, is primarily a per-I/O source of overhead. For example, in multi-client workloads dominated by small (4KB-64KB) I/Os, such as on-line transaction processing, remote memory paging [14], non-linear editing of video files, and standard office and engineering applications, performance can be limited by the server CPU, due to the per-I/O control transfer and processing overhead of RPC [34]. While overhead can be reduced with link-level and transport-level features offered by networks such as FibreChannel [18], this solution is not applicable to the widely deployed Ethernet and IP protocol infrastructure. In this paper, we explore alternative ways to reduce per-byte and per-I/O overhead in NAS systems over IP networks.

One approach to reduce per-byte overhead is to use network interface controller (NIC) support for transport protocol offload and for *remote direct data placement* (RDDP) [17]. An RDDP protocol performs network transfers directly to and from application buffers, eliminating the need for memory copying in the I/O data path. Remote direct memory access is a user-level networking [36] protocol achieving RDDP via remote memory read and write operations. The emergence of commercially-available NICs with RDMA capabilities has motivated the design of the *Direct-Access File System* (DAFS) [12,20], a network file access protocol optimized to use RDMA for memory copy avoidance and transport protocol offload. DAFS targets resource-intensive NAS applications, such as media streaming and databases.

In this paper, we argue that a simpler, alternative RDDP mechanism can offer similar memory copy avoidance and protocol offload benefits to those achieved with RDMA. This mechanism relies on *pre-posting of application buffers* at the receiver prior to the arrival of the RPC carrying the data payload [2]. This paper presents the first evaluation of a NAS system using this RDDP mechanism. Our results show that its benefits can be achieved with a kernel-based NFS client, whose two key properties are (a) support for optionally bypassing the kernel buffer cache, and (b) integration with the NIC for direct transfer to and from user-level buffers. A drawback of this approach, in contrast to the platform independent user-level client structure [20] enabled by DAFS, is that it is not as portable due to its dependence on specific kernel support.

While reduction of per-byte overhead is an important goal for NAS systems targeting I/O-intensive workloads, per-I/O overhead can limit performance of NAS servers involved in processing a large number of small I/Os issued by multiple clients. With the server CPU sat-

urated due to the overhead of interrupts, scheduling, and file processing for small I/O RPCs, the NIC data transfer engine becomes underutilized, and as a result, throughput is less than the peak achievable by the network. In addition, server CPU involvement in each RPC increases file access response time. One way to improve throughput and response time for small I/Os is to *replace RPC by client-initiated RDMA*. Client-initiated RDMA does not involve the server CPU in setting up the data transfer, and therefore, has lower per-I/O overhead on the server compared to RPC.

This paper makes the following contributions:

(a) It shows that end-system overhead reduction for NAS applications is possible with simple RDDP support on NICs offering transport protocol offload.

(b) It differentiates between *throughput-intensive workloads performing large I/Os*, which primarily depend on RDDP for copy avoidance, and *workloads performing small I/Os*, for which client-initiated RDMA is necessary to reduce server per-I/O overhead.

(c) It proposes *Optimistic RDMA*, a new network I/O mechanism that enables client-initiated RDMA and benefits workloads performing small I/Os.

(d) It evaluates *Optimistic DAFS (ODAFS)*, our extension to DAFS that uses ORDMA, to improve server throughput and response time in workloads dominated by small I/Os.

The rest of this paper is organized as follows: In Section 2, we provide background and discuss related work. In Section 3, we present the implementations of the NAS systems that use RDDP. In Section 4, we describe the design and implementation of ORDMA and ODAFS. In Section 5, we use an experimental platform consisting of a Myrinet cluster of commodity PCs to evaluate the systems discussed in this paper.

2 Background and Related Work

Network storage systems can be categorized as Storage-Area Network (SAN)-based systems, which use a block access protocol, such as FibreChannel and iSCSI, or NAS-based systems, which use a file access protocol, such as NFS. SAN-based systems preserve an important property of direct-attached block I/O device interfaces, which is the ability for direct data transfers between the communication device and a user or kernel memory buffer. However, a drawback of using a SAN to share a storage volume is the need for additional synchronization mechanisms not present in current local file systems. Additionally, storage volumes accessed by user-level applications over a SAN are not under file system control and cannot be accessed using file system tools, complicating data management. In NAS-based systems, file servers handle sharing and synchronization. In addition, NAS storage volumes are under file system management and control.

High-performance NAS applications are becoming increasingly network I/O-intensive. This is due to the emergence of servers with large memory caches and the use of aggressive file caching and prefetching policies in conjunction with powerful disk I/O subsystems. In the future, new storage technologies reducing the $/MB ratio of stable storage, such as microelectromechanical systems, or MEMS, are expected to further ease the disk I/O bottleneck. On the other hand, network hardware performance is rapidly improving, with 2-2.5Gb/s commercial implementations available today and 10Gb/s implementations expected within a year. To deliver this network performance to applications, NICs should be able to transfer data at the speed of the network link. In addition, interaction with the host should take place with minimal CPU overhead. High-performance NICs are designed to integrate DMA engines able to transfer data between host memory and the network link at hardware speeds, for both large and small (4KB-64KB) I/Os [26]. Low CPU communication overhead is possible with user-level communication libraries [26,35,36] commonly used in distributed scientific computations. NAS systems, however, are usually implemented over general-purpose network protocols, such as Ethernet and TCP/IP, and communication abstractions, such as RPC, which result in high communication overhead.

A drawback of using RPC for file I/O data transfer is that this method requires staging of the data payload in intermediate host memory buffers and copying, to move the data to its final destination. One way to solve this problem is by enabling direct data transfers between clients and storage nodes over a SAN for large I/Os, as in several emerging clustered storage systems, such as Slice [3], MPFS–HighRoad [13], NASD [15,27], GPFS [30] and Storage Tank [16]. These systems use file servers for small I/O and metadata traffic. An alternative solution that does not require a SAN is to take advantage of RDDP mechanisms applicable to RPC-based data transfer over IP networks. For example, DAFS [12,20] and NFS-RDMA [9] are two recently proposed NAS systems based on NFS and using RDMA for memory copy avoidance and transport protocol offload. This approach promises to reduce communication overhead to levels comparable to that of block channel protocols.

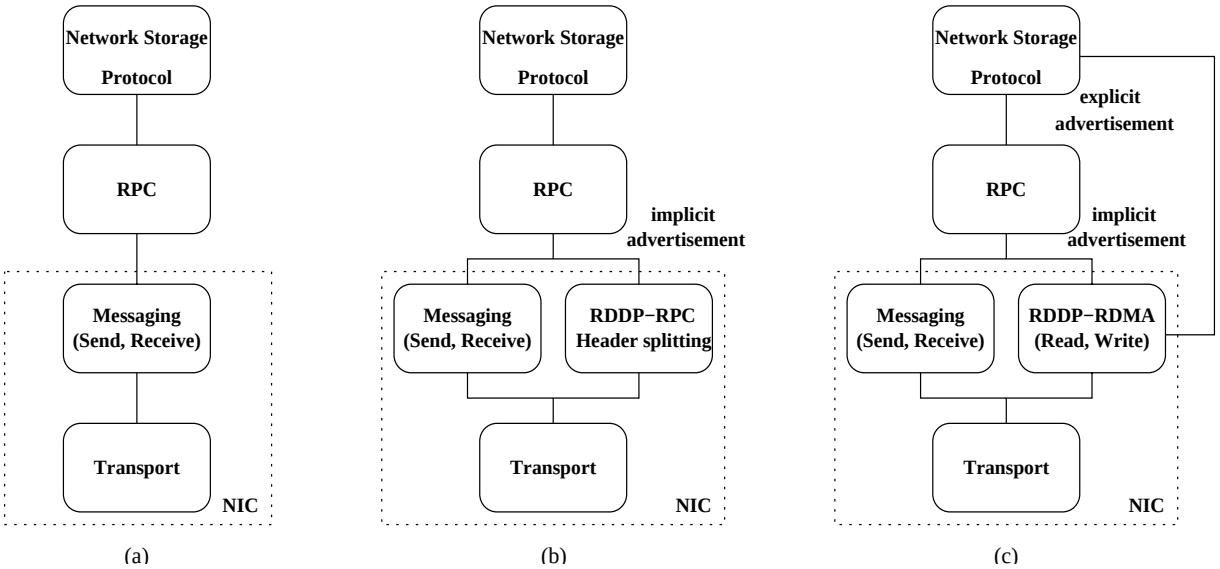

Figure 1. Protocol stack with the messaging and transport protocols offloaded to the NIC (a). RDDP is possible either by separating the data payload when in-lined in the RPC (b) or with RDMA (c).

In Section 2.1, we introduce network protocols that can be used to implement high-performance network-attached storage systems. In Section 2.2, we focus on the communication overhead of these protocols. Finally, in Section 2.3, we examine the impact of communication overhead on I/O throughput and response time.

2.1 Network storage communication protocols

Network storage systems can be implemented based on the interfaces and semantics of the network protocols shown in Figure 1. The primary communication abstraction is remote procedure call [5]. RPC can be implemented over a messaging layer, which can be offloaded to the NIC along with the transport protocol, as shown in Figure 1(a). The messaging layer can be accessed by the host via an interface that exports send and receive operations [7,35]. In addition, RDDP [17] enables direct placement of upper-level protocol data payloads into their target host memory buffers, as shown in Figure 1(b,c).

A communication layer implementing RDDP must perform the following operations: (1) Separate the protocol header from the data payload, (2) match the latter with its target buffer on the receiver, and (3) deposit it directly into its target buffer. To be able to perform (2), the target buffer must be *tagged* and *advertised* prior to the I/O. Tag advertisement can be either *implicit* or *explicit*, as shown in Figure 1, depending on whether it is performed by the RPC protocol or explicitly by the NAS protocol. In either case, however, advertisement is performed by an RPC. The data payload can be in-lined in the RPC message or transferred separately, using remote direct memory access.

RDDP using RPC (RDDP-RPC): One way to empower RPC with RDDP is to associate the target buffer with an RPC-specific tag and advertise this tag to the remote host. The remote host must include the advertised tag in the RPC that carries the data payload. The receiving NIC must match the tag with the target buffer, separate the data payload from the protocol headers (*header splitting*), and deposit the data directly into its target buffer. An RDDP-RPC mechanism evaluated in this paper is described in more detail in Section 2.2.

RDDP using RDMA (RDDP-RDMA): Another way to implement RDDP is using RDMA, which is a network data transfer protocol [8,37]. The RDMA layer exports a remote memory *read* and *write* interface. RDMA uses host virtual memory addresses as RDDP buffer tags. An RPC advertises the remote buffer and an RDMA moves the data to the target buffer. RDMA requires interaction with the upper-level protocol only to initiate the RDMA operation. It does not require interaction with the upper-level protocol at the target of the remote read or write operation. Only the RDMA initiator receives notification of completed events.

User-level networking [36] requires that RDMA use virtually addressed buffers. NICs with RDMA capabilities use a Translation and Protection Table (TPT), which is a device-specific page table, to translate virtual addresses

carried on RDMA requests to physical addresses. To avoid limiting the size of the TPT, NICs can be designed to store the entire TPT in host memory, maintaining only a TLB on-board the NIC [26,37]. Systems using RDMA need to ensure that the NIC can find virtual to physical address translations of exported pages referenced in RDMA requests and that memory pages used for RDMA are kept resident in physical memory while the transfer takes place. Page *registration* through the OS is necessary in conventional NICs on the I/O bus, to ensure that address translations are available and that pages remain resident for the duration of the DMA.

Implications of RDDP tag advertisement. Protocols using RDDP for direct data placement typically advertise buffer tags by an RPC on a per-I/O basis. Advertisement of buffer tags on a per-I/O basis, however, means that both sides are involved in setting up each data transfer. An alternative that reduces the cost of per-I/O buffer advertisement is to cache advertisements in clients and carry file access operations by RDMA only [33]. Optimistic DAFS, our extension to DAFS described in Section 4.2, uses client-initiated RDMA without requiring buffer advertisement, thereby avoiding RPCs, on each I/O.

Messaging and Transport layers. The messaging layer exports a *queue pair* (QP) interface [7,35,36] for sending and receiving messages and for event notification. The messaging layer offers data transfer and event notification only, leaving event handling to upper-level protocols such as RPC. An example of a protocol providing user-level messaging and RDMA is the Virtual Interface (VI) architecture [35]. The transport layer exports a reliable, in-order stream abstraction similar to the TCP sockets interface. In addition, transport protocol support for framing, such as in SCTP [31], is required by RDDP in order to preserve upper-level protocol header and data payload boundaries.

2.2 Communication overhead

Host communication overhead in NAS end-system hosts is defined as the length of time that the host CPU is engaged in the transmission and reception of messages [10,11,22]. It consists of a *per-byte* component $o_{\text{per-byte}}$, which is the length of time that the CPU is engaged in data touching operations such as copying or integrity checking, and a *per-I/O* component $o_{\text{per-I/O}}$, which is the length of time that the CPU is engaged in processing the I/O request incurred in network and file system protocol stacks. The *per-packet* component, due to message fragmentation and reassembly, disappears if the transport protocol is offloaded to the NIC. We will assume an offloaded transport for the remainder of this paper. The following formula expresses the client or server CPU overhead of file access in an I/O transferring m bytes:

$$o(m) = m \times o_{\text{per-byte}} + o_{\text{per-I/O}}$$

There are a number of well-known techniques [10], such as checksum offloading, interrupt coalescing and increasing the network maximal transfer unit, for reducing overhead. These techniques are offered by several high-speed NICs and supported by mainstream operating systems. Further reductions in per-byte and per-I/O overhead are possible with the network I/O mechanisms and the NAS systems described in this paper and summarized in Table 1.

Reducing per-byte overhead. The primary source of per-byte overhead is memory copying. Avoiding unnecessary memory copying is a challenging problem since it requires either significant NIC support or significant file system and network protocol stack changes, such as integration of buffering systems [29,32] or virtual memory (VM) re-mapping techniques [6]. To avoid unnecessary copying, the I/O payload should be transferred directly from the source to the destination buffer. Avoiding memory copies on the outgoing path is relatively easy using scatter/gather support at the NIC or VM page re-mapping. Avoiding copies in the receiving path is more challenging since it requires NIC support to deposit incoming data either in a page-aligned location or directly at the final destination. In this paper we consider two ways to achieve direct data placement in host memory, either within the context of RPC or in combination with RDMA:

(a) RDDP-RPC. As described in Section 2.1, the RDDP-RPC protocol, which is NAS-specific, enables the NIC to identify and separate NAS and RPC headers from the data payload and deposit the latter directly into the target buffer on the host using DMA. In our implementation, we use the RPC transaction numbers as buffer tags. A tag is associated with an application buffer at the time when the latter is *pre-posted* by the receiving host, prior to sending the RPC request. Buffer tags are *implicitly advertised* in the context of the RPC protocol message exchange. RDDP-RPC imposes no buffer size or alignment restrictions on application buffers. Pre-posting of receive buffers (or *pre-posting*, for short) has previously been used in a kernel-resident RPC-based global shared memory service [2]. In Section 3.2, we describe a NAS system based on RDDP-RPC.

Network I/O mechanism	NAS system	Uses RDMA	Per-I/O tag advertisement
RDDP-RPC (§2.2)	NFS pre-posting (§3.2)	No	Yes
RDDP-RDMA (§2.2)	NFS hybrid (§3.1), DAFS [20]	Yes	Yes
Optimistic RDMA (§4)	Optimistic DAFS (§4.2)	Yes	No

Table 1. Network I/O mechanisms and NAS systems evaluated in this paper. RDDP mechanisms target per-byte overhead. Optimistic RDMA combines RDDP and per-I/O overhead reduction.

Untagged RDDP-RPC transfers are also possible and do not require pre-posting. The data payload is placed in intermediate, page-aligned host buffers and the physical memory pages of these buffers are re-mapped into the target buffer, provided that the latter is also page-aligned. A low overhead NFS implementation using header splitting and VM page re-mapping has been evaluated in a recent study [20].

(b) RDDP-RDMA. In this method, tag advertisement is performed using RPC but data transfer is performed using RDMA, as described in Section 2.1. RDMA imposes no buffer size or alignment restrictions. In Section 3.1, we describe NAS systems using RDDP-RDMA.

Both techniques rely on transport protocol offload to the NIC. They differ, however, in the complexity of implementation and in their generality. RDMA is a general-purpose data transfer mechanism: it is independent of any NAS protocol and exports a user-level API. NICs supporting RDDP-RPC are simpler to design and implement. They are customized, however, for particular NAS protocols and export a kernel API.

Reducing per-I/O overhead. The primary source of per-I/O CPU overhead is RPC processing. The main components of RPC are event notification, either by interrupt or polling, process scheduling, interaction with the NIC to start network operations or to register memory, and execution of the file protocol processing handlers. Part of the overhead of RPC is expected to improve with advances in core CPU technology. Other parts of the per-I/O overhead, however, such as interrupts and device control, are due to the interaction between the NIC and the host over the I/O bus and therefore not expected to improve as quickly as core CPU performance.

RDMA has fundamentally lower per-I/O overhead than RPC for remote memory transfers since it does not involve the target CPU. Reducing per-I/O overhead in file clients using RDMA is possible with techniques such as *batch I/O* in DAFS [12]. Using batch I/O, a single RPC is used to request a set of server-issued RDMA operations, amortizing the per-I/O cost of the RPC on the client. Reduction of per-I/O overhead on the file server is also important, perhaps even more so since servers receive I/O load from multiple clients. Our solution to reducing server per-I/O overhead uses client-initiated Optimistic RDMA, as discussed in Section 4.

2.3 I/O throughput and response time

Throughput and response time are standard I/O metrics used to assess performance in NAS systems. In this section we describe how CPU overhead affects these metrics.

Throughput is important for applications that can sustain several simultaneously outstanding transfers, either by having some knowledge of future accesses, or by involving a number of simultaneous synchronous activities, such as concurrent transactions in OLTP. From the overhead equation of Section 2.2 and with the per-byte component of overhead associated with memory copying eliminated using RDDP, overhead is dominated by its per-I/O component.

In addition to host CPU overhead, the performance of network storage applications may also depend on other parameters [11] such as the network link latency (L) and bandwidth (BW_{network}), and the NIC transfer rate (BW_{NIC}). Modern NIC architectures using DMA engines for transfers between the network link and host memory [26] ensure that the NIC is not the bandwidth bottleneck for messages larger than a certain threshold, i.e., $BW_{\text{NIC}} > BW_{\text{network}}$.

The I/O throughput achievable with a stream of I/O requests, each of size m, can be limited either by the network or by the (client or server) CPU:

$$\text{Throughput}\ (m) = \min\left\{BW_{\text{network}}, \frac{m}{o_{\text{per-I/O}}}\right\}$$

For large I/O blocks, even a low I/O request rate can saturate the network, and the throughput is determined by BW_{network}. For small I/O blocks, however, the CPU is more likely to become the resource limiting throughput. This is because the CPU is saturated processing RPCs at lower I/O rates than necessary to keep the NIC data transfer engine fully utilized. It is therefore important to reduce the per-I/O overhead for small file accesses. A previous study found that file server throughput in NFS workloads modeled by SPECsfs is most sensitive to host CPU overhead [23].

Besides throughput, response time is also important in transactional-style network storage applications that perform short transfers and cannot hide network latency using read-ahead prefetching or write-behind policies. Such applications usually have unpredictable access patterns involving small file blocks or file attributes. Response time is the delay to satisfy a remote file I/O request and consists of the transmission round-trip time on the network link, the NIC latencies, control and data transfer costs on the host I/O buses, and interrupt and scheduling costs in the case of remote procedure call-based I/O [34]. For a heavily loaded server, response time increases by the amount of queueing delays [23].

3 Direct transfer file I/O in NAS systems

File I/O in traditional operating systems is staged in the file system buffer cache, and memory copies are usually necessary to move data between network buffers, the file system cache and application buffers. In Section 2.2, we discussed network I/O mechanisms to achieve direct data placement and avoid the cost of data movement. In this section, we examine the use of those mechanisms to implement ***direct transfer file I/O***. This differs from what is commonly referred to as *direct file I/O* and associated with the `O_DIRECT` flag of the POSIX open system call. While direct file I/O implies a disabled file cache, which does not necessarily reduce memory copying, direct transfer file I/O additionally implies copy-free data transfer between the storage device and user-space buffers. This is easily achievable in local or network-attached storage systems, over parallel or serial SCSI, by programming the disk controller to DMA the requested data blocks directly to application buffers.

Direct transfer file I/O in network file systems is more challenging, as general-purpose NICs are not aware of upper-level transport protocol packet formats and semantics and cannot usually be programmed to DMA the data payload directly into application buffers. This is possible, however, with NIC support for RDDP-RDMA or RDDP-RPC.

To take advantage of a direct transfer I/O facility, file system clients must be modified so that their I/O operations bypass the buffer cache and propagate memory buffer information to the NIC. A drawback of using direct transfer file I/O is the need to register and pin user-level buffers, as shown in Figure 2. In the case of kernel file clients, registration has to happen on-the-fly and for each I/O to be transparent to user-level applications. One problem with this requirement is the possibility that the kernel may be unable, due to per-process resource limits, to pin the user-level buffers required for the transfer. Besides introducing additional failure modes, the need for on-the-fly memory registration and de-registration introduces a performance penalty in the data transfer path.

3.1 Direct transfer file I/O using RDDP-RDMA

One way to support direct transfer I/O is with RDDP-RDMA, used in the recently proposed DAFS [12] and NFS-RDMA [9] systems. DAFS is a file access protocol [20] that performs data transfers using server-initiated RDMA read and write operations, after explicitly advertising buffer addresses using RPC. In Sun's NFS-RDMA, buffer addresses are implicitly advertised by the RPC protocol. NFS-RDMA uses client- or server-initiated RDMA read operations issued from within the RPC protocol to pull data from remote buffers.

RDDP-RDMA requires registration and pinning memory buffers on both the client and the file server. This is a disadvantage not found in RDDP-RPC, which requires registration and deregistration only on the receiving side (e.g., the client in the case of reads). An advantage of RDDP-RDMA, however, is that the frequency of host interaction with the NIC can be reduced by caching registrations at the client and the server. With RDDP-RPC, NIC interaction is required on each I/O to pre-post application receive buffers.

In Section 5.1, we evaluate the performance of a kernel-based NFS-derivative system that performs data transfers using server-initiated RDMA. Our implementation modifies the NFS wire protocol to enable remote memory pointer exchange between client and server, like DAFS, but leaves the NFS client API unchanged, like NFS-RDMA. In Section 5.1, we refer to this system as *NFS hybrid*.

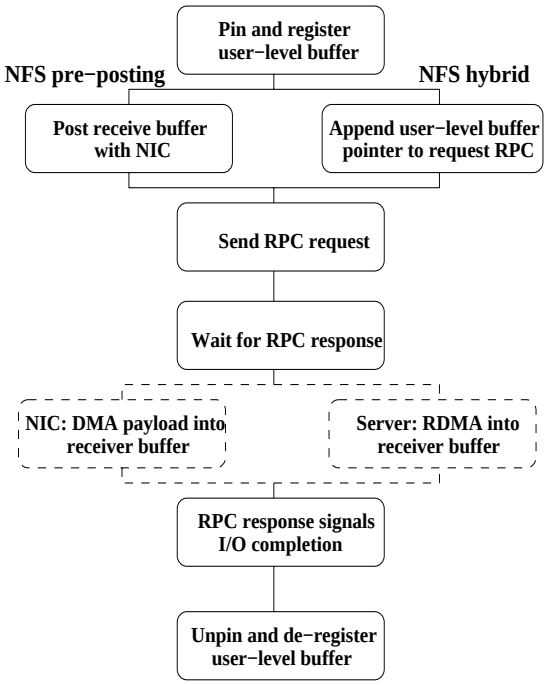

Figure 2. NFS client actions for a read request with either RDDP-RDMA or RDDP-RPC.

3.2 Direct transfer file I/O using RDDP-RPC

Another way to support direct transfer I/O is with a NIC that supports RDDP-RPC. The implementation of an RDDP-RPC-based kernel client requires a device interface that communicates the following information to the NIC:

(a) A description of the user memory buffer, including the physical address pointing to the buffer, where data coming from the network is to be directly placed.

(b) A description of the request including the RPC transaction number and the type of request, enabling the NIC to recognize the data payload in the RPC response.

This scheme requires simple modifications in the *vnode* layer of existing network file clients to avoid the user/kernel copy, pin the user-level buffer in physical memory and give the NIC the description of the user-level buffer rather than a pointer to an intermediate buffer cache location. Both synchronous and asynchronous file I/O over an NFS client offering such support enjoys zero-copy, uncached data transfer.

One drawback of this scheme is that the NIC needs to be able to parse transport and application-level headers to understand RPC responses, which raises security and safety issues. These issues can be addressed by requiring supervisor privileges to program the NIC. Another drawback is that by bypassing the buffer cache, which abstracts the device layer, the file client is no longer part of the device-independent part of the kernel. Since not all NICs are expected to support an RDDP-RPC API, the file client depends on the availability of a device-specific API. However, making NIC-assisted direct transfer file I/O a mount option is expected to work well in practice.

This paper presents the first evaluation of a NAS system using RDDP-RPC. In Section 5.1, we refer to this system as *NFS pre-posting*.

4 Optimistic RDMA

The need for buffer tag advertisement on a per-I/O basis in RDDP systems requires the use of RPCs. These RPCs contribute to per-I/O CPU overhead, reducing server throughput and increasing response time in workloads dominated by small I/Os, as discussed in Section 2.3. One way to address these problems is to use *client-initiated RDMA*, without wrapping the RDMA in an RPC to prepare the server on a per-I/O basis. In this section, we introduce *Optimistic RDMA*, a novel network I/O mechanism that enables RDMA with these properties. The following design challenges must be addressed in an ORDMA mechanism:

Ensuring safety. One way to avoid accidental corruption or malicious buffer access by mutually untrusted clients is to use cryptographically strong hashing. Each exported memory segment is associated with a *capability* [24], which is a keyed message authentication code (MAC) computed and stored at the server TPT entry for the memory segment and given to the client. A capability protecting a memory segment is sent back to the server NIC with every ORDMA request for that segment. The server NIC verifies the validity of a capability before allowing a memory access. The server may revoke access privileges to an exported memory segment, for example, when protecting or invalidating VM page translations, by locally invalidating its capability in the TPT.

Handling remote memory access faults. Client-initiated RDMA may be faced with a number of exception conditions at the target NIC. For example, some of the targeted VM pages may no longer be resident in physical memory. In addition, targeted pages may be locked or protected. In the case of non-resident pages, one option is to enable the NIC to trigger a page-in disk I/O. However, this solution significantly increases the com-

plexity of the NIC design and most importantly, it may not be supported by the OS. The ORDMA model enables clients to initiate RDMA that is guaranteed to succeed only if the target buffer is valid and exported by the server and is neither locked nor protected. In the opposite case, a recoverable access fault is signaled to the client by a network exception. After catching an ORDMA exception, a client handler may recover by retrying the access using an alternate access method, such as RPC.

Two important design choices in any ORDMA-based system are: (a) how a client finds references to server memory buffers, and (b) how a client handles exceptions due to failed ORDMAs. Section 4.2 describes the choices we have made in the Optimistic Direct Access File System.

4.1 ORDMA implementation

The two main ORDMA implementation issues are (a) how to synchronize between the NIC and the host CPU when accessing VM pages, and (b) how to report NIC–to–NIC network exceptions in case of remote memory access faults.

NIC–host CPU synchronization in accessing VM pages. Synchronization is necessary because the NIC is allowed to set up DMA transfers between the network and main memory, independently of the CPU. The kind of NIC–host CPU synchronization depends critically on OS support for multiple processors. An ORDMA-capable NIC in a multiprocessor OS can fully participate in the VM system, by pinning/unpinning and locking/unlocking VM pages in response to network events. This is because a multiprocessor OS offers the necessary synchronization structures for the NIC to appear indistinguishable from an additional CPU to the OS, except for its performance. On the other hand, a NIC in a uniprocessor OS may not be able to pin pages from interrupt handlers if, for example, the OS is non-preemptive. In this case, synchronization via the host memory resident TPT is necessary.

The NIC should ensure that the following two conditions hold for the duration of DMA: First, pages involved in DMA have to remain resident in physical memory. Second, conflicting accesses by another CPU or NIC should not be allowed. We chose to satisfy both requirements by treating VM pages with translations loaded in the NIC TLB as both pinned and locked. The alternative of locking pages only for the duration of an I/O requires frequent NIC–host CPU interaction and was deemed too expensive in the case of a NIC on the I/O bus. All pages in the TPT, except those with translations loaded on the NIC TLB, may be locked and invalidated by the host. The NIC updates the state of TPT entries by interrupting on each TLB miss. These interrupts increase CPU overhead but have the side-effect of speeding up the loading of TPT entries into the NIC, which is now done via a host-initiated programmed I/O operation, instead of (possibly several) NIC-initiated DMA on the PCI bus.

A drawback of having to synchronize via a device-specific page table is that the OS has to be aware of and adapt to the idiosyncrasies of the NIC. For example, it should always check with the NIC TPT before reclaiming a page and account for the fact that attempts to reclaim a physical page may fail until the page is evicted from the NIC TLB. To avoid starvation, the OS must increase its minimum free page threshold by the maximum amount of physical memory with page translations loaded on the NIC TLB. The OS must also be able to limit the effective size of the NIC TLB to avoid excessive pinning by the NIC.

NIC–to–NIC exceptions. ORDMAs may fail due to a variety of conditions, such as invalid address translation, protection violation, failure to lock page(s). We decided to support such exceptions by extending the VI protocol with recoverable RDMA failure semantics. Since VI is a layer on top of Myrinet's GM in our prototype, we first modified the Myrinet GM Control Program to report such conditions as exceptions in low-level *get* (i.e., RDMA read) and *put* (i.e., RDMA write) operations. These exceptions are reported as "soft" or recoverable transport errors in the VI descriptor status flags, and can be appropriately handled by higher-level software, such as the DAFS client and the ODAFS user-level cache described in Section 4.2.1.

4.2 Optimistic DAFS

The Optimistic Direct Access File System is our extension of the DAFS [12] protocol. Just like DAFS, ODAFS can use RPCs for all file requests. In addition to RPC requests, ODAFS clients may issue ORDMAs to directly access exported data and metadata buffers in the server file cache.

ODAFS is based on the following key principles:

(a) Clients maintain a *directory* or cache of remote references to server memory. These directories can be built either *eagerly* when clients ask the server for memory references, or *lazily* when the server piggybacks memory references with each RPC response.

(b) Directory entries need not be eagerly invalidated when the server invalidates VM mappings for exported references. Instead, invalid ORDMAs are caught at the server NIC, which throws exceptions reported to clients. An important advantage of this consistency mechanism is that the server does not need to keep track of clients caching memory references.

(c) The client is always prepared to catch an exception for each ORDMA operation. In such a case, the client issues an RPC to access the data.

Other important considerations for ODAFS clients are determining the size of the ORDMA directory, particularly in relation to the memory requirements for file data and attribute caching, and the replacement policies appropriate for maintaining the ORDMA directory. In this paper, we assume that the size of the ORDMA directory is small compared to the size of the data cache, and use the LRU replacement algorithm for ORDMA references. However, since ORDMA accesses are expected to be issued in response to client cache misses, a more appropriate strategy would be similar to the multi-queue algorithm for storage server caches [38].

4.2.1 ODAFS implementation

We implemented prototypes of an ODAFS client and server by extending the following existing DAFS components: a user-level DAFS file cache [1], a user-level DAFS API implementation [20] and a DAFS kernel server [21]. We rely on the ORDMA support for Myrinet described in Section 4.1.

The ODAFS server piggybacks remote memory references to data blocks in its kernel file cache onto RPC responses to the client. The ODAFS client stores these references in cache block headers. As data blocks are reclaimed by the client cache, memory references are allowed to live in "empty" headers. The client cache is configured with many more empty headers than data blocks. Ideally, it should have enough buffer headers to be able to map the entire server physical memory available for file caching.

We also modified the DAFS API to allow passing of ORDMA references, and the DAFS client implementation to include ORDMA operations in its event loop. On ORDMA exceptions, the DAFS client retries the operation using RPC in order to guarantee success. At RPC completion, the fresh piggybacked reference to the server buffers is passed to the ODAFS client.

The ODAFS server maps file blocks on a private 64-bit virtual address map. This is to ensure that there is always enough virtual address space to map large amounts of physical memory for long periods of time. Thus, we ensure that NIC TLB invalidations are due to the OS reclaiming a VM page due to memory pressure and never due to having to share a small virtual address space. This 64-bit address space is addressable only by the NIC and never by the CPU. It is therefore independent of whether the CPU has a 32- or 64-bit architecture.

Ideally, the replacement algorithm used in the server NIC TLB should be the same as the algorithm used in the client ORDMA directory.

4.2.2 Benefits and limitations

ODAFS is targeted for workloads performing small I/Os. ODAFS is most beneficial with significant memory-to-memory I/O traffic, such as that caused by small files and attribute accesses, and high server cache hit rates. The benefit comes mainly from the low server CPU overhead of the ORDMA mechanism. However, there are a number of workload characteristics that limit the applicability of ORDMA, and consequently the effectiveness of ODAFS. These are:

Few remote memory accesses, e.g., when client caching is effective in locally satisfying most file requests [25]. Note that this factor reduces the usefulness of any remote file access protocol.

Low ORDMA success rate, i.e., low server cache hit rates. If many ORDMAs result in failure, ODAFS performance is similar to that of DAFS as the cost of ORDMA exceptions and subsequent RPCs is masked by the high latency of server disk I/O.

Many file accesses that cannot be satisfied via ORDMA. This could be because the remote memory location of the target data may not be exportable. Examples are directory name lookups, which require significant processing on the server besides the actual data transfer.

Small read–write ratio. Writes require the update of associated file state, such as time of last modification and file block status on the server, besides the actual data transfer. Append-mode writes are harder as they further require allocating disk blocks on the server, checking resource limits, and potentially serializing over concurrent appending accesses.

Low NIC TLB hit rates. Satisfying TLB misses for a NIC on the I/O bus can be significantly more expensive than for a CPU TLB. In addition, network storage working sets can be very large and access patterns may not have enough locality to render NIC TLBs effective.

Finally, mixing ORDMA- and RPC-based file access has implications on the atomicity of file I/O. RPC-based file access guarantees that the entire I/O operation is atomic by locking the entire file for the duration of the I/O. However, ORDMA-based file access guarantees that at most one memory word is read or written atomically. By using both access methods, ODAFS effectively offers ORDMA's atomicity semantics. For UNIX file I/O semantics, client applications should explicitly lock files for the duration of I/O.

5 Experimental Results

Our experimental setup consists of a cluster of four PCs each with a 1GHz Pentium III processor, 2GB SDRAM and the ServerWorks LE chipset. The PCs are connected via a 2Gb/s switch over full-duplex ports. Each NIC has a 200MHz LANai9.2 network processor with 2MB of on-board SRAM in 64MHz/66-bit PCI slots. PCI bus throughput is measured at 450MB/s. All PCs run FreeBSD 4.6. The LANai drivers and firmware are based on GM-2.0 alpha1 release featuring support for remote direct memory access *get* and *put* primitives. The VI library is based on the Myricom VI-GM 1.0 release. This is a host-based user-level library mapping VI operations to GM operations and used by the user-level DAFS client [20]. A kernel port of the VI library supports the DAFS/ODAFS server [21]. Ethernet emulation is implemented in the standard LANai GM-2.0 firmware and drivers and supports UDP and IP checksum offloading and interrupt coalescing. The Ethernet packet MTU is 9KB. GM data transfers, however, are fragmented and reassembled by the LANai using a 4KB MTU. The GM driver and firmware are modified as described in Section 3.2 for RDDP-RPC and Section 4.1 for ORDMA (except for capabilities, which are not yet supported in our implementation). *NFS pre-posting* and *NFS hybrid* are implemented by modifying the FreeBSD 4.6 kernel, as shown in Figure 2. *NFS pre-posting* uses the RDDP-RPC device interface. *NFS hybrid* uses GM *put* to perform server-initiated RDMA writes to client memory buffers. Given the very low transmission error rates of Myrinet, we use UDP as our transport protocol to avoid the higher overhead of TCP. This configuration approximates the benefits of offloading TCP if it were supported by the NIC. Table 2 reports baseline network performance of the protocols used over the Myrinet network. These numbers are collected using the *gm_allsize*, *pingpong* and *netperf* programs for GM, VI-GM and UDP/IP protocols respectively.

Protocol	Roundtrip (us)	Bandwidth (MB/s)
GM	23	244
VI	23 poll	244
	53 block	244
UDP/Ethernet	80	166

Table 2. Baseline Myrinet performance. One-byte roundtrip time.

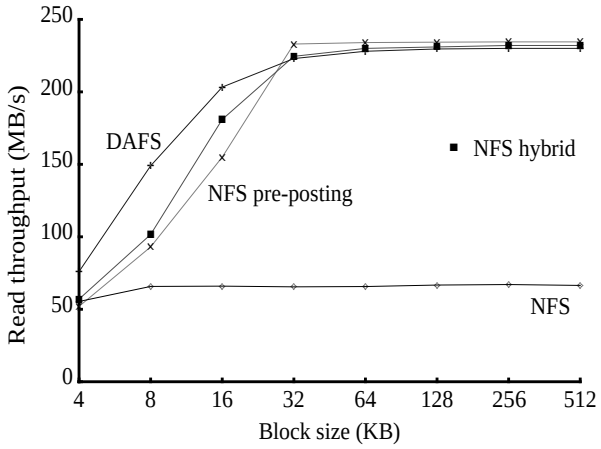

Figure 3. Client bandwidth performing read-ahead with variable application I/O block size.

5.1 Client overhead

In this section, we measure read throughput with a simple client and application performance with the Berkeley DB database.

Client read throughput. This experiment measures file read throughput with a simple client performing asynchronous read-ahead without any data processing. We compare DAFS to the two optimized NFS implementations, *NFS pre-posting* and *NFS hybrid*, and to standard NFS. The client reads data sequentially, using a varying block size, from a 1.5GB file warm in the server file cache. Read-ahead prefetching at the application level is done via the DAFS and POSIX *aio* APIs. NFS is mounted with the readahead parameter set to zero in all cases. UDP/IP is modified so that the NFS transfer size can match the application block size up to 512KB.

Figure 3 shows that for block sizes larger than 32KB DAFS can sustain read throughput of about 230 MB/s. As shown in Figure 4, it achieves this throughput consuming less than 15% of the client CPU for 64KB or larger blocks, by offloading the transport to the NIC and by being able to avoid all memory copies. Per-I/O overhead is progressively better amortized since the unit of data movement always matches the application block

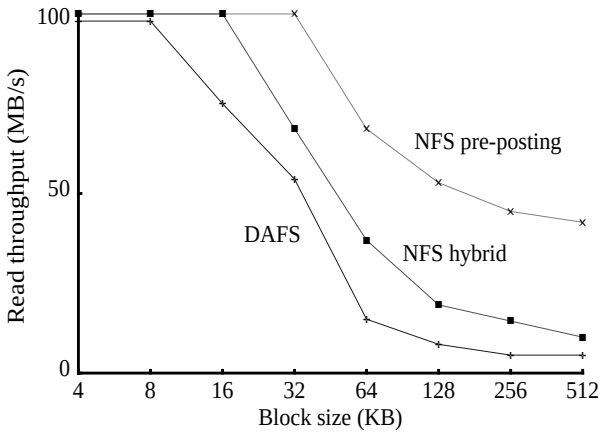

Figure 4. Client CPU utilization performing read-ahead with variable application I/O block size.

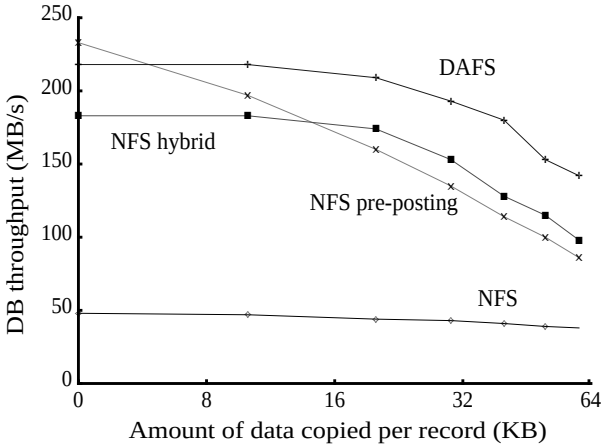

Figure 5. Berkeley DB performing asynchronous I/O.

size. For small block sizes, DAFS achieves low per-I/O overhead by using polling instead of interrupts. Similarly to DAFS, *NFS hybrid* sustains 230 MB/s for block sizes of 32KB or larger with CPU utilization dropping exponentially with increasing block size. However, even though both DAFS and *NFS hybrid* use RDMA, *NFS hybrid* uses more of the client CPU due to its higher per-RPC overhead. Both DAFS and the *NFS hybrid* clients avoid registering application buffers with the NIC on each I/O by caching registrations.

NFS pre-posting sustains 235 MB/s for block sizes 32KB or larger, performing data transfer in 8KB IP fragments. It slightly outperforms systems using RDMA because the size of Ethernet packets (8KB) is twice the size of the 4KB GM fragments. The decline in its client CPU utilization is eventually limited for large block sizes as the total number of IP fragments is independent of the block size. In addition, the *NFS pre-posting* client interacts with the NIC for pre-posting application receive buffers on each I/O. Standard NFS (not shown in Figure 4) achieves a maximum throughput of 65 MB/s, limited primarily by memory copying, which saturates the client CPU.

Berkeley DB performing asynchronous I/O. In this experiment, we use Berkeley DB to show the effect of client CPU overhead in application performance. Berkeley DB [28] (*db*) is an embedded database management system that provides recoverable, transaction-protected access to databases of key/data pairs. It is linked into the application address space and maintains its own user-level cache of recently accessed database pages. *Db* is modified to asynchronously prefetch database pages when it is possible to pre-compute a set of required pages.

In this experiment, an application uses *db* to compute a simple equality join with 60KB records. The result of the join is a large list of keys, retrieved from the database file located on the server. *Db* pre-computes the list of required pages and performs read-ahead, maintaining a window of outstanding I/Os. To vary the computational requirements of the application, we increase the amount of data copied from the *db* cache into the application buffer for each record, from one byte to 60KB, and report the application throughput in Figure 5. The throughput sustained by the application when there is little memory copying is close to the wire throughput for all systems except standard NFS. *NFS pre-posting* performs slightly better than the other systems, as is also the case in Figure 3. As the amount of copying increases, performance becomes limited by the client CPU. Relative system performance is inversely proportional to each system's client CPU overhead for 64 KB network I/O transfers.

5.2 Server I/O throughput and response time

In this section we present microbenchmark and PostMark results highlighting the properties of ORDMA and the upper bounds for performance improvements in ODAFS applications. In all cases, a file cache based on DAFS open delegations [12] is interposed between the application and the DAFS/ODAFS API. To avoid introducing platform-specific parameters, such as the cost of NIC memory registration and TLB misses, we ensure that RDMA is done on pre-registered buffers and always hits in the NIC TLB. The cost of a NIC TLB miss is about 9µs for ORDMA in our prototype. This penalty can be reduced in NICs that have large TLBs, are integrated on the memory bus, or share a TLB with the host CPU [4].

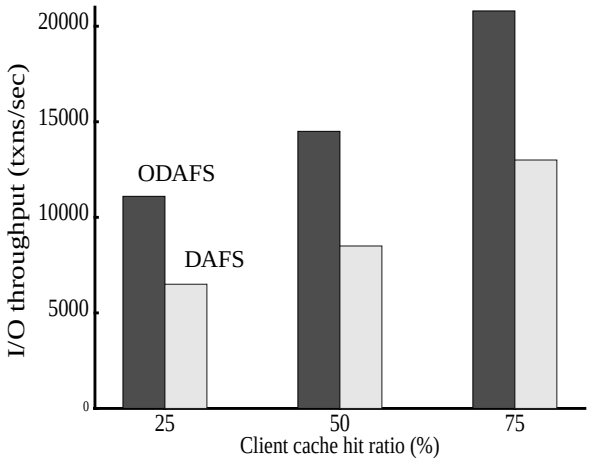

Figure 6. PostMark I/O throughput. Single client with variable cache hit ratio.

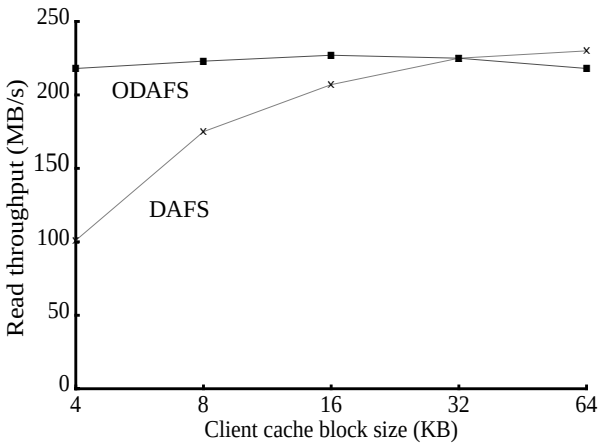

Figure 7. Server throughput. Two clients reading a large file using a large block size.

Microbenchmarks. We measure I/O response time in reading a 4KB block from server memory using (a) in-line RPC read, that is, the data payload in-lined with the RPC response, (b) direct RPC read, that is, the data payload transferred by server-initiated RDMA write, and (c) client-initiated ORDMA read. The file cache is configured with a small number of data blocks but with a large number of headers that can retain remote memory references. In this microbenchmark, a simple application sequentially reads a 1GB file warm in the server cache twice, in increments of 4KB. The client cache is configured with a 4KB block size and is cold prior to starting the experiment.

I/O mechanism	Response Time (us)	
	in mem.	in cache
RPC in-line read	128	153
RPC direct read	144	144
ORDMA read	92	92

Table 3. I/O response time with 4KB block size.

During the first pass, all I/O requests miss in the client cache, which, in response, initiates remote file accesses using either in-line or direct RPC. RPC responses carry remote memory references to file blocks on the server cache. During the second pass, I/Os still miss in the client cache. However, this time remote I/O may also be performed by ORDMA since the client cache managed to map the entire file on the server after having accessed it once during the first pass. Table 3 shows the I/O response time during the second pass using different network I/O mechanisms. RPC in-line involves a memory copy in the client from the communication buffers to the file cache. ORDMA yields about 36% lower response time than direct RPC.

Effect of client caching. In this experiment, we model a file client accessing a set of small files synchronously over DAFS and ODAFS. The file set size exceeds the client cache size in all cases. We model such a latency-sensitive workload by configuring the PostMark [19] benchmark for read-only transactions without file creations or deletions. Each read I/O is preceded by a file open and followed by a file close operation. After the first open of a file, which grants the client an open delegation, each subsequent open or close for that file is satisfied locally. We use a 4KB average file size and configure the client cache with a 4KB block size. The client cache hit ratio determines the frequency of remote memory access. By varying the size of the client cache and keeping the file set size constant we progressively increase its hit ratio from 25% to 50% to 75%. We find that in all cases ODAFS yields about 34% higher throughput than DAFS (Figure 6), reflecting the difference in response time between ORDMA and direct RPC. This is because, despite the benefit of client caching, overall performance is sensitive to the cost of remote memory accesses. The DAFS server CPU utilization drops from 30% to 25% to 20% as the client cache hit ratio improves. However, ODAFS uses no server CPU after it manages to collect remote memory references for the entire server cache, which occurs after the client has accessed each file at least once.

Server throughput. In this experiment, we show the effect of per-I/O overhead on server throughput. We model a multi-client, throughput-intensive workload dominated by small I/Os by configuring two clients to sequentially read a 1GB file warm in the server cache twice, using a large block size. For reads larger than the

cache block size, the cache starts internal read-ahead up to the size of the application request. To vary the unit of network I/O, we progressively increase the cache block size from 4KB to 64KB and measure server throughput for each cache block size during the second pass, as shown in Figure 7. We find that with ODAFS, the two clients are able to saturate the server network link for all cache block sizes (except for 64KB due to a performance bug in GM *get*) without using the server CPU. DAFS yields lower server throughput for small I/O blocks, saturating the server CPU due to processing direct RPCs. For the smallest cache block size of 4KB for which the difference between DAFS and ODAFS is maximal, the DAFS server is primarily constrained by network interrupts. Switching to polling for all network events, DAFS throughput improves to about 170 MB/s reducing the performance improvement attainable from ODAFS to 32%.

6 Conclusions

In this paper, we show that two network I/O mechanisms for RDDP, *pre-posting application receive buffers* and *RDMA*, are effective in reducing per-byte CPU overhead in NAS end-systems. Our experiments show that they both enable a throughput-intensive streaming client to achieve file access at the speed of a 2Gb/s network link. RDMA offers the advantage of a general-purpose user-level API, enabling portable user-level implementations. Workloads dominated by small I/Os are more sensitive to per-I/O overhead. For such workloads, we propose a new network I/O mechanism, *Optimistic RDMA*, that aims to improve server throughput and response time. We have implemented a prototype of ORDMA and of *Optimistic DAFS*, our extension of the DAFS protocol that uses ORDMA. We measured improvement in server throughput and response time by up to 32% and 36%, respectively, in small I/O transfers.

7 Acknowledgements

This research was supported by Network Appliance and Sun Microsystems. We thank our shepherd Peter Corbett, Dan Ellard, Omri Traub, Matt Welsh, and the anonymous reviewers for their helpful comments. We also thank Andrew Gallatin of Myricom Inc. for his help with Myrinet software.

8 Software Availability

All NFS, DAFS and ODAFS software, including FreeBSD patches and Myrinet driver, library and firmware modifications, used in this paper is freely available from http://www.eecs.harvard.edu/dafs.

References

[1]. S. Addetia, “User-level Client-side Caching for DAFS”, Harvard University TR-14-01, March 2002.

[2]. D. Anderson, J. Chase, S. Gadde, A. Gallatin, K. Yocum, “Cheating the I/O Bottleneck: Network Storage with Trapeze/Myrinet”, in *Proc. of USENIX Annual Technical Conference*, pp. 143-154, New Orleans, LA, June 1998.

[3]. D. Anderson, J. Chase, A. Vahdat, "Interposed Request Routing for Scalable Network Storage", in *Proc. of 4th USENIX OSDI Symposium*, pp. 259-272, San Diego, CA, October 2000.

[4]. B. Ang, D. Chiu, L. Rudolph, Arvind, “Message Passing Support on StarT-Voyager”, *CSG Memo 387*, MIT Laboratory for Computer Science, July 1996.

[5]. A. Birrell, B. Nelson, “Implementing Remote Procedure Calls”, *ACM Transactions on Computer Systems*, (2)1:29-59, Feb. 1984.

[6]. J. Brustoloni, “Interoperation of Copy Avoidance in Network and File I/O”, in *Proc. of the IEEE INFOCOM’99 Conference*, pp. 534-542, New York, NY, March 1999.

[7]. P. Buonadonna, D. Culler, “Queue-Pair IP: A Hybrid Architecture for System Area Networks", in *Proc. of 29th ISCA Symposium*, Anchorage, AK, May 2002.

[8]. G. Buzzard, D. Jacobson, M. Mackey, S. Marovich, J. Wilkes, “An Implementation of the Hamlyn Sender-Managed Interface Architecture”, in *Proc. of 2nd USENIX OSDI Symposium*, pp. 245-259, Seattle, WA, October 1996.

[9]. B. Callaghan, NFS over RDMA, *Work in progress presented at 1st USENIX FAST Conference*, Monterey, CA, January 2002.

[10]. J. Chase, A. Gallatin, K. Yocum, “End System Optimizations for High-Speed TCP”, *IEEE Communications*, (39)4:68-74, April 2001.

[11]. D. Culler, R. Karp, D. Patterson, A. Sahay, K. Schauser, E. Santos, R. Subramonian, T. von Eicken, “LogP: Towards a Realistic Model of Parallel Computation”, in *Principles and Practice of Parallel Programming*, pp. 1-12, 1993.

[12]. M. DeBergalis, P. Corbett, S. Kleiman, A. Lent, D. Noveck, T. Talpey, M. Whittle, “The Direct Access File

System", to appear in *Proc. of 2nd USENIX FAST Conference*, San Francisco, CA, March 2003.

[13]. EMC Cellera HighRoad, White Paper http://www.emc.com/pdf/proucts/celerra_file_server/HighRoad_wp.pdf, January 2002.

[14]. E. Felten, J. Zahorjan. "Issues in the Implementation of a Remote Memory Paging System", CS TR 91-03-09, University of Washington, March 1991.

[15]. G. Gibson, D. Nagle, K. Amiri, J. Buttler, F. Chang, H. Gobioff, C. Hardin, E. Riedel, D. Rochberg, J. Zelenka, "A Cost-Effective, High-Bandwidth Storage Architecture", in *Proc. of 8th ASPLOS Conference*, pp. 92-103, San Jose, CA, October 1998.

[16]. D. Pease, IBM Storage Tank, *Work in progress presented at 1st USENIX FAST Conference*, Monterey, CA, January 2002.

[17]. IETF *Remote Direct Data Placement* (RDDP) Working Group, http://www.ietf.org/

[18]. C. Jurgens, "FibreChannel: A Connection to the Future", *IEEE Computer*, 28(8):88-90, August 1995.

[19]. J. Katcher, "PostMark: A New File System Benchmark", Network Appliance TR-3022, October 1997.

[20]. K. Magoutis, S. Addetia, A. Fedorova, M. Seltzer, J. Chase, D. Gallatin, R. Kisley, R. Wickremesinghe, E. Gabber, "Structure and Performance of the Direct Access File System", in *Proc. of USENIX Technical Conference*, Monterey, CA, pp. 1-14, June 2002.

[21]. K. Magoutis, "Design and Implementation of a Direct Access File System Kernel Server for FreeBSD", in *Proc. of USENIX BSDCon 2002 Conference*, San Franscisco, CA, pp. 65-76, February 2002.

[22]. R. Martin, A. Vahdat, D. Culler, T. Anderson, "Effects of Communication Latency, Overhead and Bandwidth in a Cluster Architecture", *Proc. of the 24th Annual ISCA*, pp. 85-97, Denver, Colorado, June 1997.

[23]. R. Martin and D. Culler, "NFS Sensitivity to High-Performance Networks", in *Proc. of SIGMETRICS '99/ PERFORMANCE '99 Joint International Conf. on Measurement and Modeling of Computer Sys*., pp. 71-82, Atlanta, GA, May 1999.

[24]. S. Mullender and A. Tanenbaum, "The Design of a Capability-based Distributed Operating System", *The Computer Journal*, 29(4):289-299, 1986.

[25]. D. Muntz and P. Honeyman, "Multi-level Caching in Distributed File Systems (your cache ain't nuthin' but trash), In *Proc. of USENIX Technical Conference*, pp. 305-314, San Antonio, TX, January 1992.

[26]. Myricom LANai9.2 and GM communication library, Myricom Inc., http://www.myri.com

[27]. D. Nagle, G. Ganger, J. Butler, G. Goodson, C. Sabol, "Network Support for Network-Attached Storage", in *Proc. of Hot Interconnects*, Stanford, CA, August 1999.

[28]. M.Olson, K. Bostic, M. Seltzer, "Berkeley DB", in *Proc. of USENIX Technical Conference (FREENIX Track*), pp. 183-192, Monterey, CA, June 1999.

[29]. V. Pai, P. Druschel, W. Zwaenepoel, "IO-Lite: A Unified I/O Buffering and Caching System", in *Proc. of 3rd USENIX OSDI Symposium*, pp. 15-28, New Orleans, LA, February 1999.

[30]. F. Schmuck and R. Haskin, "GPFS: A Shared-Disk File System for Large Computing Clusters", in *Proc. of 1st USENIX FAST Conference*, Monterey, CA, January 2002.

[31]. R. Steward, C. Metz, "SCTP: New Transport Protocol for TCP/IP", IEEE Internet Computing, pp. 64-69, November 2001.

[32]. M. Thadani, Y. Khalidi, "An Efficient Zero-copy I/O Framework for UNIX", SMLI TR95-39, Sun Microsystems Lab, Inc., May 1995.

[33]. C. Thekkath, H. Levy and E. Lazowska, "Separating Data and Control Transfer in Distributed Operating Systems", *6th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 2-11, San Jose, CA, October 1994.

[34]. C. Thekkath and H. Levy, "Limits to Low-Latency Communication on High-Speed Networks", *ACM Trans. on Computer Systems*, 11(2):179-203, 1993.

[35]. Virtual Interface Architecture Specification, Version 1.0, http://www.viarch.org, December 1997

[36]. T. von Eicken, A. Basu, V. Buch and W. Vogels, "U-Net: A User-Level Network Interface for Parallel and Distributed Computing", *Fifteenth ACM Symposium on Operating Systems Principles*, pp. 40-53, Copper Mountain Resort, CO, December 1995.

[37]. M. Welsh, A. Basu and T. von Eicken, "Incorporating Memory Management into User-Level Network Interfaces", in *Proc. of Hot Interconnects*, pp. 27-36, August 1997.

[38]. Y. Zhou, J. Philbin, K. Li, "The Multi-Queue Replacement Algorithm for Second Level Buffer Caches", in *Proc. of USENIX Technical Conference*, pp. 91-104, Boston, MA, June 2001.